Scalable Characteristic Mode Analysis: Requirements and Challenges

Khulud Alsultan[1], Praveen Rao[2], Anthony Caruso[1], Ahmed Hassan[1]

[1]University of Missouri-Kansas City, [2]University of Missouri-Columbia

Characteristic Mode Analysis (CMA) decomposes the total current of a scatterer in terms of its fundamental modes or eigen-currents. Moreover, it provides the modal significance of each mode at a particular frequency and also provides the modal excitation coefficient, which can be interpreted as the radiation characteristics of each mode. Therefore, CMA can be used in explaining the response of scatterers with complex shapes in a wide range of applications that include antenna design, nano-electromagnetics, and electromagnetic compatibility and interference. Recently, CMA has been used for the analysis of a wide range of scatterers that include worm-like carbon nanotubes, crumpled graphene flakes, aircrafts, and unmanned aerial vehicles (UAVs). In many of the previous cases, the scatterers are highly complex in shape or are electrically large yielding a Method of Moments (MOM) impedance matrix that contains thousands of rows and columns. Such a large MOM impedance matrix requires gigabytes of disk space and RAM memory for storage and analysis. For example, a 30K x 30K matrix of complex numbers (i.e., $a + bi$) is 16GB in size (in binary). For CMA, the MOM impedance matrix needs to be evaluated at every frequency. Therefore, if an application requires CAM on millions of frequencies given matrices with (thousands of rows and columns) to accurately quantify the electromagnetic response, TBs of RAM and storage as well as high speed networking are needed creating a "huge data" problem.

Harrington's method [1] is a widely accepted and accurate method for CMA. It involves a sequence of matrix operations to compute the eigenvalues and eigenvectors. It includes computing singular value decomposition (SVD), multiplication, inverse, slicing, and transpose of matrices. On large matrices, the storage, transfer, and processing of these matrices will demand new advances in computer networking, storage technology, and cluster computing with generous number of hardware accelerators (e.g., GPUs, FPGAs). In this white paper, we discuss the requirements and the challenges for enabling scalable CMA for next-generation applications such as counter UAV defense systems. We also present a pilot study using commodity hardware.

Requirements/Challenges:

- *Data transfer:* A single matrix with 25 billion complex numbers is very large in size. Transferring this matrix from a machine into a compute cluster is slow especially when dealing with thousands of matrices. Hence, high speed networking is critical for CMA.
- *Computation:* Fast processing of matrix operations is indispensable for CMA. Among the different matrix operations, SVD is the slowest. Although parallel implementation of SVD is available in Apache Spark[1], complex numbers are not supported. Even on a real-valued matrix of size 16K x 16K, Apache Spark required 4 hours to compute the SVD using 32 nodes on CloudLab [3]. Certain hardware accelerators can provide a promising solution to speed up matrix operations.
- *Storage:* CMA on large matrices produces several intermediate matrices that need to be written to storage. Writing into a distributed file system such as Hadoop Distributed File System (HDFS) is a viable option.

[1] http://spark.apache.org

Pilot Study Using Harrington's Method [1,2]:

We conducted a pilot study to speed up CMA on a single machine using NumPy[2], CuPy[3], and TensorFlow[4]. (Complex numbers are supported by these software packages.) We used a machine on CloudLab with 2 Intel Xeon processors (8-core CPU, 3.2GHz), 128GB RAM, two 1TB SSDs, and NVIDIA 16GB Tesla V100 GPUs. Our CMA implementation on TensorFlow was slow when using only CPUs. It took 163 mins for a 30K x 30K matrix due to TensorFlow's inherent lack of exploiting multicores for SVD. Our CMA implementation on TensorFlow with GPU acceleration was problematic on large matrices (30K x 30K and beyond) because TensorFlow allocated GPU memory for the entire lifetime of the process. This caused out-of-memory errors after a few matrix operations. However, our CMA implementation using NumPy and CuPy was promising. NumPy exploited multicores for matrix operations. SVD was always performed on multicore CPUs. CuPy enabled all other matrix operations to be run on the GPU. The below table shows the wall-clock time taken for CMA on different matrices. For instance, it took around 18 mins to perform CMA on a 30K x 30K matrix using a CPU-GPU implementation compared to 29 mins with only CPUs. Thus, hardware accelerators can significantly improve CMA performance.

Matrix size (NxN)	File size (in binary)	Using multicore CPUs (NumPy)	Using multicore CPUs + GPU (NumPy/CuPy)
2208 x 2208 ≈ 2K x 2K	76MB	0m 8.07s	0m 9.69s
4776 x 4776 ≈ 4K x 4K	350MB	0m 8.99s	0m 13.32s
16,608 x 16,608 ≈ 16K x 16K	4.2GB	3m 12.21s	2m 37.07s
33,024 x 33,024 ≈ 30K x 30K	16.2GB	28m 39.16s	18m 29.80s

We plan to investigate how cluster computing can further improve CMA performance. One immediate next step is to use a hyperconverged infrastructure with 10's of GPUs, TBs of flash memory storage and RAM, and gigabit networking. Such an infrastructure would cost a few million dollars. We hope federal agencies will invest in such experimental testbeds for advancing computational electromagnetics in general.

References

[1] Harrington, R., and J. Mautz. "Computation of characteristic modes for conducting bodies." *IEEE Transactions on Antennas and Propagation* 19.5 (1971): 629-639.

[2] Alsultan, K., Rao, P., Caruso, T., and Hassan, A. "Scalable Characteristic Mode Analysis Using Big Data Techniques." In *2019 International Symposium on Electromagnetic Theory* (EMTS 2019), San Diego, 1 page, CA, 2019.

[3] Duplyakin, D., Ricci, R., Maricq, A., Wong, G., Duerig, J., Eide, E., Stoller, L., Hibler, M., Johnson, D., Webb, K., and et al., "The design and operation of CloudLab." In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, 114 (2019).

[2] https://numpy.org/
[3] https://cupy.chainer.org/
[4] https://www.tensorflow.org/